INTENSIFICATION OF SOFTWARE SECURITY USING SECURE OBFUSCATION WITH INJECTING AND DETECTING CODE CLONES

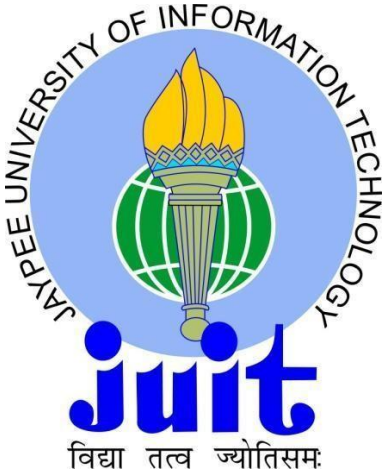
Synopsis submitted in partial fulfillment of the requirements for the Degree of

DOCTOR OF PHILOSOPHY

BY

PRATIKSHA GAUTAM

Enrollment No. 136206



Under the Supervision of

Dr. Hemraj Saini

Department of Computer Science and Engineering JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY
Waknaghat, Solan-173234, Himachal Pradesh, INDIA

October, 2018

ABSTRACT

Software security plays a vital role in Information Technology as it's infringe lead to tremendous monetary depletions due to the resale of application thus; it is a potential research issue in the present scenario. It can be classified as coding errors and copyright protection of software. There are three types of copyright protection attacks as software piracy, reverse engineering, and tampering. The security approaches against these assaults are software watermarking, code obfuscation and tamper-proofing. The code obfuscation is one of the pioneer methods of software protection against reverse engineering; obfuscate a code into a form which is thornier for an attacker to comprehend or modify the original code from the illegal reverse engineering procedure. The code transformation assaults can be categorized as 1) static analysis, 2) dynamic analysis, and 3), code clone attack. The reverse engineering attacks occur due to the unconfined software code to the user. Malicious reverse engineering of software codes can be compact by the travail of code obfuscation on source code. None of the existing code transformation technique protects from reverse engineering assaults. A novel code transformation method is needed for software security. Therefore, to avert static analysis assault and dynamic analysis attack on obfuscation and to make code compact for an adversary. Farther, the code clone (non-trivial software clone) is used to thwart static and dynamic analysis attacks on obfuscation. The code clone detection is another emerging research issue in software clone detection area because of it is considered adverse in software evolution and maintenance. Though various software clone detection tools and techniques have been proposed in the literature, however, in spite vigorous study there is conspicuous inadequacy of exertion in the identification and study of type-1 to type-4 software clones. Although, software clones can be classified by their features as textually equivalent software clone or functionally identical code clones. Furthermore, syntactic equivalent clones are categorized into type-1 to type-3 while semantically equivalent code clones are type-4. The type-1 software clones are identical code fragments excluding few amendments in whitespaces, blank spaces, comments, etc. while type-2 is similar to original segments except renaming identifiers. Type-3 clones are those where to insert or delete new lines in the source code, and type-4 software clones have similar functionality with the different structure. Thus, in this thesis firstly a state-of-art in software security as well as in software clones detection and their analysis in various manners. Foremost, to proposed hybrid approach which is based on tokenization concept and it can detect type-1(similar code fragments with some adoptions as whitespaces, blank space, comments, etc.) with high precision and recall. Subsequently, proposed an approach for the detection of type-2(renaming identifiers with comments, white spaces, etc.) software

clones with the help of a directed acyclic graph which is a compiler optimization technique.

Moreover, the proposed approach is considered to deal with the distinct types of software clones using the proposed editing mutation operator-based taxonomy. Third, we develop a method "for the detection of trivial- software clones by using an approach namely as a control flow graph (CFG) and reduced flow graph (RFG). The non-trivial software clones detected by using program dependency graph (PDG)". Fourth, to evaluate the performance of clone detection tools and techniques from two perspectives. Foremost, to evaluate regarding software metrics and subsequently, by generated test cases. Fifth, a mutation operators-based editing nomenclature for replica generation that replica developers' expurgation behaviors in the copy/pasted code in a top-down manner. Subsequently, by using proposed taxonomy, we accomplished a scenario-based qualitative assessment and evaluation of the existing clone tools and approaches for evaluating software clone detection tools and techniques. Furthermore, by using the result of this research, the users can choose right tool according to their requirements as well as it shows the constraints of a peculiar approach. Sixth, in order to evaluate software clone detection tools and techniques in rational manner and to evade challenge and manual massive endeavor for injecting software clones in source code as well as for validating detection tools and techniques, we have proposed an automatic mutation operator-based evaluation framework that automatically injects type-1 to type-2 code clones in the program text. Furthermore, it measures the detection tools precision and recalls with the minimum threshold value. This study has revealed that proposed approach is adequate for the detection of type-1 to type-3 software clones which are created with the help of mutation operator-based editing classification in large-scale application system with high precision and recall.

Table of Contents

	PAGE NO.
ABSTRACT	i-ii
Table of Contents	
1. INTRODUCTION	1-6
1.1 Motivation and Contributions	2-6
1.2 Organization of thesis	6-6
2. Enhancement of Software Security using Obfuscation and prevention of attacks on Obfuscation	7-9
2.1 Securing against static attack	7-8
2.2 Securing against dynamic attack	8-9
2.3 Summary	9-9
3. Detection of software code clones	10-14
3.1 Detection of Type-1 Code Clones using Hybrid approach	10-11
3.2 Detection of Type-2 Code Clones using DAG	11-11
3.3 Detection of Type-3 Code Clones using CFG and RFG	12-13
3.4 Summary	13-14
4. Performance evaluation of existing code clone detection tools and Techniques	14-16
4.1 Evaluation of clone detection tools by software metrics	14-15
4.2 Performance evaluation of software clone detection tools by generated Test-cases	15-15
4.3 Summary	16-16
5. Uses of mutation operators in code clone	16-21
5. 1 Proposed Mutation Operator based editing taxonomy	16-17
5. 2 Mutation operator-based Scenario for evaluating Software Clone detection tools and techniques	17-19
5.3 An automatic framework for injecting and detecting software clones	19-20
5.4 Summary	20-21
6. CONCLUSION AND FUTURE WORK	21-22
7. REFERENCES	23-25
8. LIST OF PUBLICATIONS	26-27

1. INTRODUCTION

Software security is a potential research issue in present's scenario due to the expansion in digital technology because of the risk of illegal copying of software. Although, the increasing piracy rate has posed a severe threat to software developers leading to the development of various software protection techniques. Software security can be classified into two types as coding errors and copyright protection which are shown in figure 1.

Figure 1 shows the coding errors and copyright protection. The coding errors are further classified into three types as genesis means how the problem entered the system. Time means at which point in the production cycle problem entered and the last is location; it means where the problem is manifest [1]. The genesis-based errors classified into two types as intentional and inadvertent while intention again divided into two parts as malicious and non-malicious. Time-based errors classified into three types as maintenance, operation and development. Development farther classified into three types as design, object code and source code. At last location-based error can be classified into two types as software and hardware. The software-based error can be categorized into application, support system, and operating system [1]. However, there are a number of software protection techniques have been developed against various malicious attacks as software piracy, reverse engineering, and tampering which are illustrated in figure 1. The defense approaches against these attacks are software watermarking, code obfuscation and tamper-proofing [2-3]. Code obfuscation aims to change a program into a form which is thornier for an attacker to comprehend the functionality of the code. The primary motive for securing software through obfuscation arises from the three types of reverse engineering attacks which are mentioned below:

- Static analysis attack: statically analysis of data.
- Dynamic analysis attack: to check code on distinct inputs.
- Code clone detection attack: to identify copied code [4-7].

The code clone detection has become a prominent research issue in software engineering, over the past few years due to the replication of code segments in the source code. Reusing source code fragments from one location and paste them into another place with significant alteration is code clone and the pasted section is copied code.

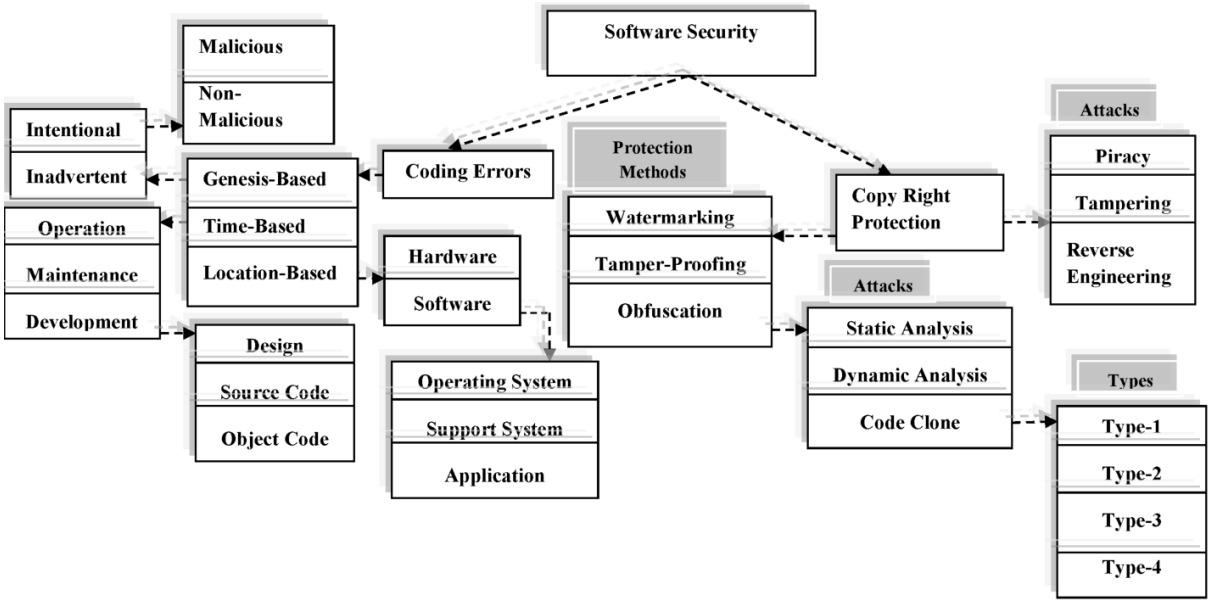


Figure 1 Graphical representation of problem formulation

The software clones can be classified by their aspects as syntactic similarity-based software clones or functionality similarity-based software clones. The syntactic similarity-based clones are identical in their structure as well as in functionality. The textual/syntactic similarity-based clones can be classified as type-1 to type-3, and semantic or functional similarity-based clones are type-4 which is illustrated below:

- Type-1: Identical code fragment except for variations in white space, comments, etc.
- Type-2: Syntactically similar code segments except alterations in function names or variables.
- Type-3: Identical code fragments with some variation as statements added and deleted.
- Type-4: identical functionality distinct structure [8-10]. However, there are several code clone detection approaches have been developed for the detection of type-1, type-2 and type-3 software clones and each technique has its pros and cons. But none of the techniques detects type-1, type-2 or type-3 software clones with some potency benchmark as scalability, portability, robustness, precision, recall. Therefore, there is a need to develop an approach which can detect type-1 to type-3 clones with an effectiveness criterion.

1.1 MOTIVATION AND CONTRIBUTION

Software piracy, the illegal copying, and resale of applications, is a 62.7 billion dollar in 2013

and more than 400 billion dollars from cyber attacks in 2016[11]. The recent report of BSA [11] and joint research which is accomplished by National University of Singapore and IDC, ISTR [12, 13] presents that malicious code is found in the pirated software. The survey report also illustrates per year piracy rate as shown in table 1 and that due to this pirated software we have spent millions of dollar per year for software protection which is shown in table 2. Thus, malware are a big concern in the software industry and pirated software, or unlicensed software is one of the causes of malware. Therefore, there is a need for copyright protection of software. Though, many software protection techniques have been developed for copyright protection as software watermarking, obfuscation and tampering [2-3]. The obfuscation is one of the protection methods of reverse engineering which protects source code from reverse engineering attacks by making code harder for an adversary [4-6]. The reverse engineering attacks have three types as the static attack (it comprehends the functionality of software by statically analyzing program), dynamic attack (to run software with different inputs) and code clone attack (to detect and remove the duplicate code from source code) [7]. Therefore, the first research objective focuses on securing code obfuscation from static and dynamic attacks using code clone attack. The second objective focus on code clone attacks; in code clone attack duplicate codes have been inserted by copying/pasting activity in source code segments during the software development. The copying and pasting are known as software cloning and copied code is called duplicated code.

Table 1: 2016 BSA Piracy Study (Piracy Rate)

S. No	2015	2013	2011	2009
India	58%	60%	63%	65%
Total AP	61%	62%	60%	59%
Central and Eastern Europe	58%	61%	62%	64%
Latin America	55%	59%	61%	63%
Middle East and Africa	57%	59%	58%	59%
North America	17%	19%	19%	21%
Western Europe	28%	29%	32%	34%
Total World Wide	39%	43%	42%	43%
European Union	29%	31%	33%	35%
BRICS Countries*	64%	67%	70%	71%

BRIC Countries* are Brazil, Russia, India, South Africa and China.

To demonstrate the scalability confronts created by the new use-cases, study a retail banking software system sustained by Tata Consultancy Services (TCS), which is in active use by several banks (with distinct codebases).

Table 2: 2016 BSA Piracy Study Result (Retail Revenue) ($M)

S. No	2015	2013	2011	2009
India	$2,684	$2,911	$2,930	$2,003
Total AP	$19,064	$ 21,041	$20,998	$16,544
Central and Eastern Europe	$3,136	$5,318	$6,133	$4,673
Latin America	$5,787	$8,422	$7,459	$6,210
Middle East and Africa	$3,696	$4,309	$4,159	$2,887
North America	$10,016	$10,853	$10,958	$9,379
Western Europe	$10,543	$12,766	$13,749	$11,750
Total World Wide	$52,242	$62,709	$63,656	$51,443
European Union	$11,060	$13,486	$14,433	$12,469
BRICS Countries*	$14,452	$17,187	$17,907	$14,453

BRIC Countries* are Brazil, Russia, India, South Africa and China.

The TCS determines a universal codebase for all these banks. Thus, the main inspiration being complicated due to the business losses which occur due to duplicated attempts in (i) conveying general factors and (ii) retaining these general factors individually. To begin with, being one of the threats the organization suffered was to identify the blocks of common code across such large codebases. Every codebase executes into million lines of code that extends beyond only few thousand COBOL programs, most of all are about 100K LOC. The complete report of this case study is present in [14-15]. Thus, the primary objective of Tata Consultancy Services (TCS) is to detect an entire identical section of code across whole codebases to assist the group forms conclusions s about creating the general code base. Please consider that while we endorse that with the current evolutions, there are several tools and techniques for software clone detection, the previous real-life state of affairs only focus how important it is to enhance the research in this way for it to be helpful in the industrial setting. Though several clone detection techniques and tools have been developed, but few of the tools have been implicated in an observable estimation, and various efforts have been made to assess and evaluate numerous state-of-the-art tools methodically. Furthermore, a little experimental valuation has been performed by measuring the performances of these tools. The existing study demonstrates that there are a lot of facets that could influence the authenticity of the results of these assessments [16-18]. To overcome the effect of this facet; there is a need to introduce an outline of the mutation operators-based nomenclature of distant kinds of software clones so that on the basis of proposed classification several clones can be generated. The code clone is a result of some alterations in the source code, and these modifications are carried out by some specific rules which are called mutant operators and the resulting faulty versions are called mutation generation [2-3]. Some

amending transformation can generated the flawed variants in source code, and these mutation operators used to create probable flaws in code clones to transform the original program [19-27]. The proposed taxonomy used to generate scenarios which can empirically valuation of software clone detection approaches and tools. The software clones usually occur in source code due to copying from one section of source code and pasting them into another place of code. The process of copying/pasting is called software cloning and duplicated code is called a software clone. A bug identified in one place of source code requires amendments in entire replicated segments of source code. Various scientists have reported 66% [28-32] source codes copied due to copying/pasting. The problem with the duplicated code is that it increases the efforts need to be done when enhancing the source code. The software clones can be categorized as syntactic and semantic software clones. The syntactic software clones further classified as type-1(similar code segments except for some variations in comments, whitespaces, etc.), type-2(identical code segments excluding some modifications in variable's names, identifiers etc) type-3(similar code fragments with further alterations; adding, deleting statements). The semantic clones can be assorted as type-4 clones (distinct syntactic structure but similar functionality; also known as non-trivial clones) [7-8]. There are several software clone detection techniques have been existed in the literature which can detect type-1, type-2, and type-3 software clones, with fewer effectiveness criteria (precision, recall, portability, scalability, robustness). Hence, there is a need to develop an approach which can detect type-1 to type-3 and non-trivial software clones with effectiveness criteria. However, there is also requires some faulty version or duplicated code in software clone detection to evaluate software clone detection approaches. Several researchers consequently have taken the methods of introducing flaws into the correct program to generate faulty variants. These bugs can be pioneered by hand or generated automatically through a faulty version of program text. Typically, we study an automatically-generated variant as a result of applying some editing activities in the source code. Thus, there is a need to develop a tool for injecting software clones in the source code automatically. Therefore, the contribution to secure obfuscation and detection of software clones is briefed in the following steps:

- Ensuring the security of obfuscation from static and dynamic analysis attacks.
- Use of mutation operators for creating an editing taxonomy for software clones as well as to generate the mutation operator based scenario for evaluation of clone detection tools and techniques.

- Detection of type-1 to type-3 software clones as well as non-trivial software clones.
- A mutation operator-based framework for injecting software clones in software code automatically.

1.2 ORGANIZATION OF THE THESIS

The organization of the thesis is divided into eight chapters as given below. The introduction, motivation of work, contribution of the thesis, problem statements and outline of the thesis are reported in CHAPTER 1. This chapter presents a short description to the thesis. After motivating this research, it states the problem. Further, it lists the contribution of the thesis. In addition to this, the research subareas with their problem statements are also covered in this chapter. CHAPTER 2 evidences the background and preliminaries for the research in this thesis. This chapter details the software security definition, its protection methods as software watermarking, tamper-proofing, and obfuscation, and their attacks such as piracy, reverse engineering and tampering. In addition to this, there are three types of attacks against obfuscation as static analysis, dynamic analysis and code clone is also discussed with open research issues. CHAPTER 3 deals with the software clone research area. This chapter elaborates the software clones definitions with its pros and cons, software clone applications, and its detection process is briefed. The software clone detection techniques, tools with their properties are also elaborated in this chapter. CHAPTER 4 deals with the need of security in obfuscation from static and dynamic analysis attacks. The secure approach is proved by showing the simulation results. CHAPTERS 5 discuss the detection of software clones (type-1 to type-3) from the source code and it further concise the outcomes for the proposed method with proper results. CHAPTERS 6 present the evaluation of clone detection tools and techniques in two perspectives; first clone detection tools evaluation has been performed on the basis of software metrics and subsequent performance evaluation has been compared on the basis of generated test cases by software metrics as precision, recall, portability, scalability, and recall. CHAPTERS 7 presents the uses of mutation testing in software clone detection research era and it also provides an automatic framework for evaluating software clone detection tools and techniques. Finally, CHAPTER 8 concludes the thesis that comprises the thesis summary, concluding remarks, the contribution of the work done and suggestions for the future scope in this field.

2. SECURING OBFUSCATION AGAINST STATIC ENHANCEMENT OF SOFTWARE SECURITY USING OBFUSCATION

To provide software protection against malicious reverse engineering attacks of the software code, there is a need for a peculiar code transformation (semantic-preserving transmutation of a program) approach for software protection. Therefore, the first objective is to secure a program P by using obfuscation as a form which impedes the perceptibility of the program text and generates a new program (P'). The transformed code is semantically equivalent as an original program [27-28], the obfuscation's security is outlined in terms of three reverse engineering attacks as 1) Static analysis (recapitulate information about a code by analyzing). 2) Dynamic analysis (to execute code on different inputs and evaluate their outputs). 3) Code clone to identify duplicated code from the original program to comprehend the functionality of the software. The major obscurity in designing\ an adequate code obfuscator is to guarantee security; i.e.to proves that no algorithm can violate software protection in real time. Although several developed code transformation tools and techniques rely on the informal notion of protection and it can't be considered as provably secure. The most of the software developer's concern is to develop a pioneer and enhanced obfuscation technique over intellectual property. It has been reported by various researchers as well as the software developers [33-35] that conclusive obfuscation, in general, is imprudent.

2.1 PREVENTING AGAINST STATIC ANALYSIS BY AN ADVERSARY

Although, there are various code transformation methods [36] and tools present for obfuscating programs. Here, we are illustrating a few of the obfuscating techniques and tools. Kulkarni & Metta [7] have proposed a non-trivial code-clone based obfuscation approach for software protection; though, this method enhances the cost and diminishes execution speed. Ceccato [37] analyzes the key approaches of obfuscation as identifier renaming which enlarges an adversary's attempt to comprehend a given program. The control flow obfuscation based approach for software protection is used by Low [38]. The program's logic is hiding by control flow transformation with the help of fake control flow. Collberg et al., [39] have proposed boolean-valued expressions, opaque predicates, whose value is well-known to an obfuscator but it is

unable to infer the conclusion of an automatic deobfuscator. The authors [40] have exhibits a more sophisticated control flow flattening method. The proposed approach can protect static analysis while [5] shows that it was unable against hybrid (static and dynamic) attacks. (Collberg et al., [41] and Kulkarni et al., [7] demonstrate a different obfuscation approach in which source code was improved by code clones while Cohen, 1987 [42] detailed that it is improbable for an attacker to single out two code segments which are semantically identical. Nonetheless, none of the existing techniques and tools accredits extension of approaches based on software clones and methods which thwart against static and dynamic analysis attacks. While the current methods have some constraints in respect of performance overhead. The performance overhead acquaints an additional cost for each transformation in perspective of execution time and memory usage essential to perform the transformed code as well as no long-term security. Thus, the main purpose of the proposed approach is to obstruct reverse engineering attacks as well as to protect software, without alteration of the intrinsic functionality of the software. The flowchart of the corresponding proposed mechanism is depicted in figure 2.

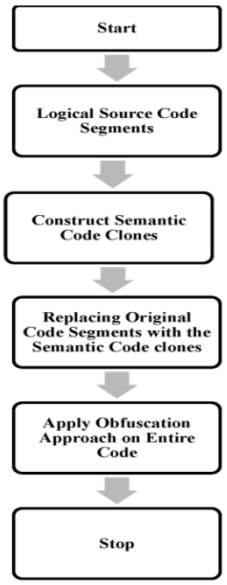

Start
Logical Source Code Segments
Construct Semantic Code Clones
Replacing Original Code Segments with the Semantic Code clones
Apply Obfuscation Approach on Entire Code
Stop

Figure 2 Flowchart of the proposed approach

2.2 PREVENTING AGAINST DYNAMIC ANALYSIS BY AN ADVERSARY

The proposed approach protects software against static analysis attacks and increases the

complexity of dynamic analysis attacks for an adversary. In dynamic attack, adversary runs source code on different inputs and analysis their outputs. Therefore, the proposed approach is based on non-trivial software clones (they have a different structure but perform similar computations). The non-trivial software clones increase the complexity for an attacker by making software intricate. However, the comparison between the proposed approach and the existing approach is shown in figure 3. The proposed method is implemented on open source java projects as Gantt project and open source java obfuscator tools as Dan's tool, Daft logic and Packers tools. The performance of the proposed approach is compared with the existing method in perspective of file size and execution time. The figure 3 depicted the graphs of the proposed mechanism in comparison of the existing method (kulkarni's obfuscation).

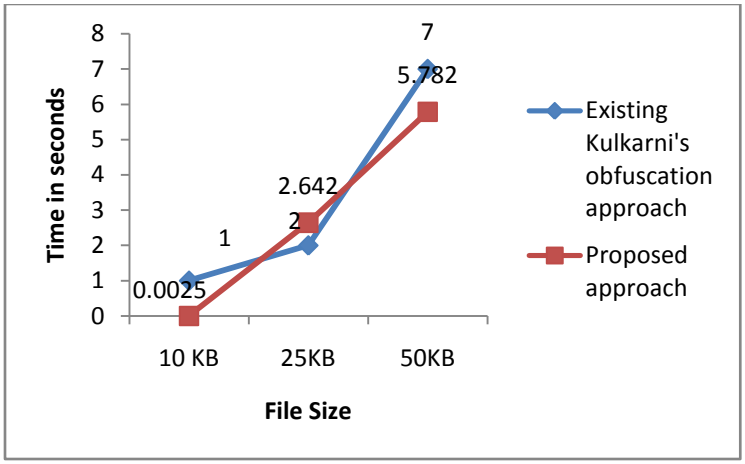


Figure 3 Comparisons of Proposed Approach with Existing Kulkarni Obfuscation Approach

2.3 SUMMARY

In this chapter, we have focused on providing an obfuscation approach for software protection. The proposed approach resists against reverse engineering attacks as static and dynamic attacks and overcomes the drawbacks of the existing technique. The technical contribution of the paper is described as follows, 1) Non-trivial software clones used to protect against static attack, and 2) Increase the complexity of dynamic attack, 3) To increase execution speed and reduce cost or enhance the efforts for an attacker.

3. DETECTION OF SOFTWARE CODE CLONES

The software clone occurs due to replication in the source code, replicated code called software clones and the process is called software cloning. The software cloning activity enhances software maintenance efforts as well as bug propagation due to copying and pasting. The entire source code requires correction if a bug is detected in one section of source code. The software clones can be classified into four types as type-1 to type-4 on the basis of their attributes such as textual similarity-based (type-1 to type-3), Functional similarity-based (type-4). The previous literature reveals that there is some code clone detection techniques have been anticipated. Further, each method has some limitations as some of the methods detect only type-1 clones with low effectiveness criteria (precision, recall, scalability, robustness, portability). Hence, there is a need to develop an approach which can detect type-1 and type-2 software clones with high effectiveness criteria. This chapter has been consists of three section as section 3.1 proposed a method which can detect type-1 software clones and type-2 software clone detection method has been discussed in section 3.2. The section 3.3 presents type-3 code clone detection approach.

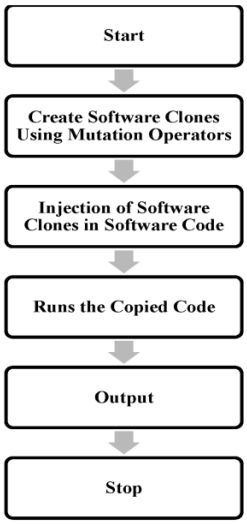
3.1A HYBRID APPROACH FOR DETECTION OF TYPE-1 CODE CLONES

The type-1 software clones have the similar syntactic structure with the same functionality while semantic clones have the different structure but similar functionality. Hence, this chapter introduced an approach which can detect type-1 software clones with high effectiveness criteria. Foremost, software clones generated by mutation-operators and subsequent a hybrid approach used to identify type-1 software clones.

Figure 4 depict the proposed method which is based on tokenization notion in which whole source code is divided into tokens, and it removes all the whitespaces comments (type-1 software clones generated by using mutation operators) etc. Although, the proposed approach is also compared with the existing approach on the basis of precision, recall, are shown in figure 5.

Type-2 software clone detection has been detected by using a directed acyclic graph. The proposed approach is based on the directed acyclic graph. At first, software clones are generated by using mutation operators and then convert source code into three address code. Further, by using three address codes, a directed acyclic graph is generated. The flow chart of the proposed

approach is shown in figure 6 and figure 7 illustrates the comparisons of proposed method with the existing method (Bauhaus tool).



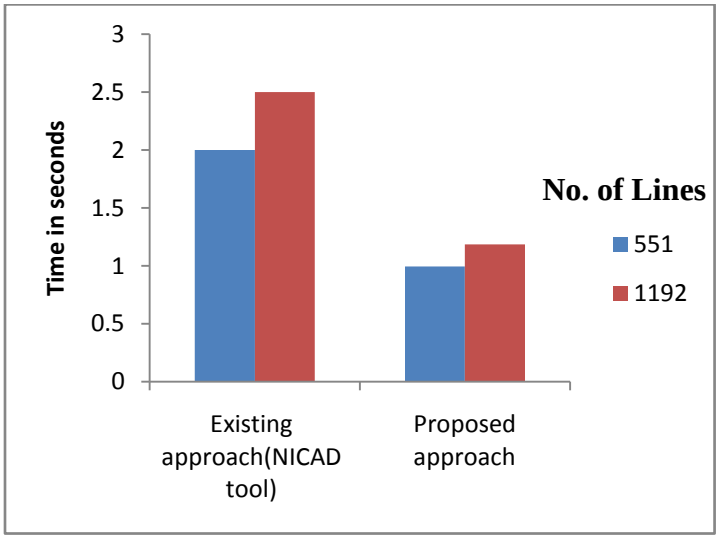
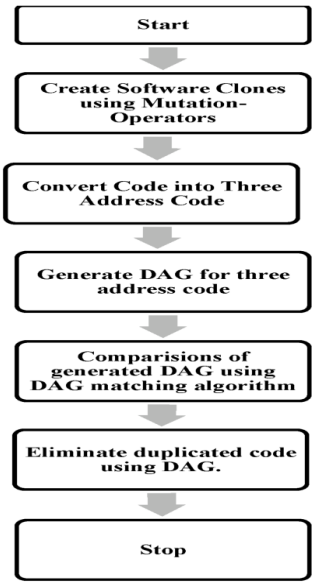


Figure 4 Flow chart of proposed approach

Figure 5 Comparison of proposed approach with existing approach(NICAD tool)

3.2 DETECTION OF TYPE-2 SOFTWARE CLONES USING DAG


Start
Create Software Clones using Mutation-Operators
Convert Code into Three Address Code
Generate DAG for three address code
Comparisions of generated DAG using DAG matching algorithm
Eliminate duplicated code using DAG.
Stop

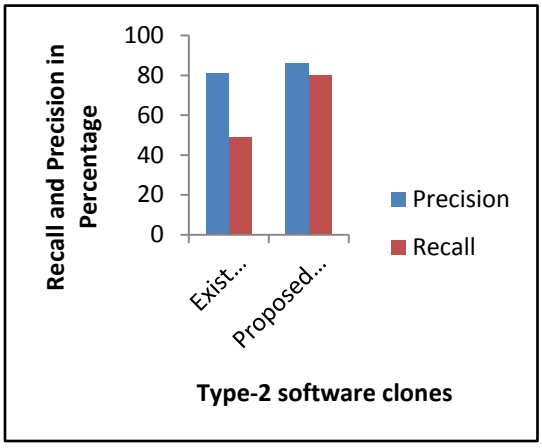


Figure 6 Flow chart of proposed approach

Figure 7 Comparisons with existing approach

3.3DETECTION OF TYPE-3 SOFTWARE CLONES USING RFG AND CFG

Reusing code segments from one section of the source text and pasting them into another section with minor modifications is called software clone and the process is called software cloning. The software clones can be classified on the basis of their features as textual identical–based and functionally identical-based. The textually similar software clones categorized into three types as type-1(similar code segment except for some variations in white spaces, comments, etc.) type-2(identical code segments except some variants in renaming identifier) type-3(identical code segments except some alterations in an insert, delete statements) type-4(syntactic dissimilar but semantically similar). There are numerous software clone detection methods, and tools have been introduced in the literature in which some of them were used to detect syntactically, and few of the approaches were used to identify semantic software clones. Thus, to overcome this issue, the manuscript proposed two approaches for syntactic (trivial) and semantic (non-trivial) software clones. The proposed method is based on two methods as a control flow graph, reduced flow graph, and program dependency graph. The program dependency graph (PDG) used to detect semantic clones or non-trivial software clones (similar functionality but different structure) while syntactic software clone detects by using control flow graph (CFG) and reduced flow graph (RFG). PDG is a hybrid approach which consists of control dependency and data dependency and the data is presented by edges while vertices consequent to statements and predicates. The data dependence presents the data flow relationship of a program and control flow relationship is represented by control dependence which is derived from the control flow graph. Figure 8 shows the flow chart of the proposed approach.

The proposed can efficiently spawn a control flow graphs and by using the generated control flow graph and reduced flow graph approach, the trivial software clone, an identical textual structure, can be identified. The proposed method is based on a tokenization conception in which whole source code is partitioned into lexemes. Figure 9 demonstrates the comparisons of proposed approach with the existing techniques.

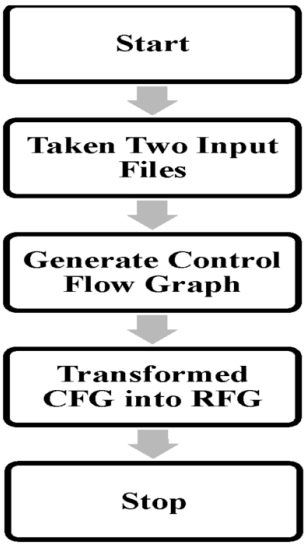


Figure 8 Flow chart of proposed approach

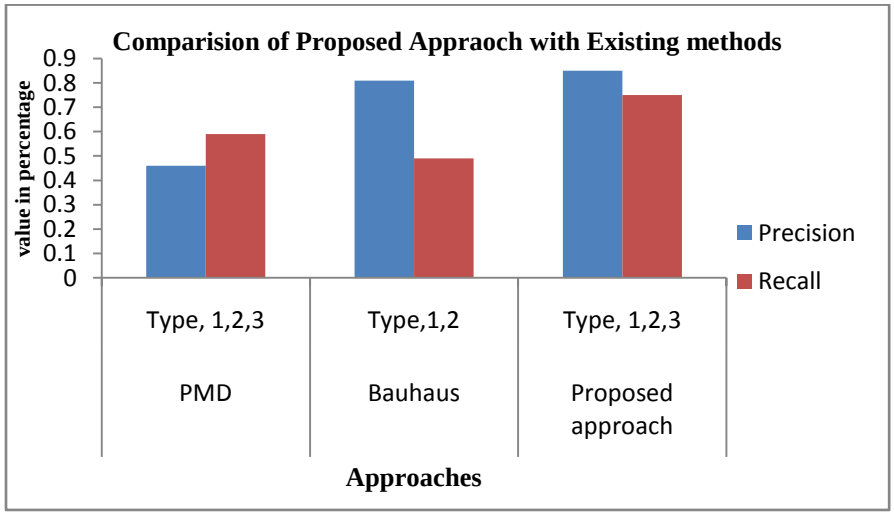


Figure 9 Comparison of Proposed Approach with Existing Methods

3.4 SUMMARY

This chapter has proposed three approaches for the detection of type-1, type-2, and type-3 code clones. The type-1 software clones detected by using a hybrid approach. The hybrid approach consists of two methods as string method and token-based method. The Results of The hybrid approach is compared with existing approaches. The second approach used directed acyclic graph (DAG) for the detection of type-2 software clones. The DAG is a compiler optimization

method for common elimination sub-expressions from the source code. Moreover, both proposed approaches also used mutation operators-based editing taxonomy for generating the maximum number of software clones. Furthermore, this chapter also focused on detection of type-3/trivial code clones and non-trivial software clones. The non-trivial software clones detected by using an existing static analysis technique while on the other side trivial/type-3 code clones identified by using a proposed approach which is used to generate control flow graph of source code. Moreover, by using the generated CFG of source code as well as of duplicated code a new approach named as reduced flow graph (RFG) has been used to compare both CFGs for the detection of type-3 code clones.

4. PERFORMANCE EVALUATION OF EXISTING CODE CLONE DETECTION TOOLS AND TECHNIQUES

Over the past few decades, many tools and methods have been proposed by many scientists to identify clone automatically in a program/software. Though, it is not specified yet how to estimate clone detection tools in perspective of accuracy, scalability, and portability. Nevertheless, all of the tools have some pros and constraints, but the usage of these tools rely on the user requirements so it is essential for the users that they should be well aware of the tools and its distinguishable features. This paper shows the performance of six clone detection tools regarding accuracy, scalability, and portability which is shown in figure 10. The primary objective of this research is to form election of tools simple for detection of duplicated code.

4.1EVALUATION OF CLONE DETECTION TOOLS BY SOFTWARE METRICS

This paper presents the evaluation of existing clone detection tools and techniques in two perceptions. Foremost, clone detection tools have been assessed by software metrics such as precision, recall, portability, scalability, robustness as shown in figure 10 and subsequent detection tools have been evaluated on the basis of generated test-cases shown in figure 11. Figure 10 shows the overall performance of software clone detection tools by precision.

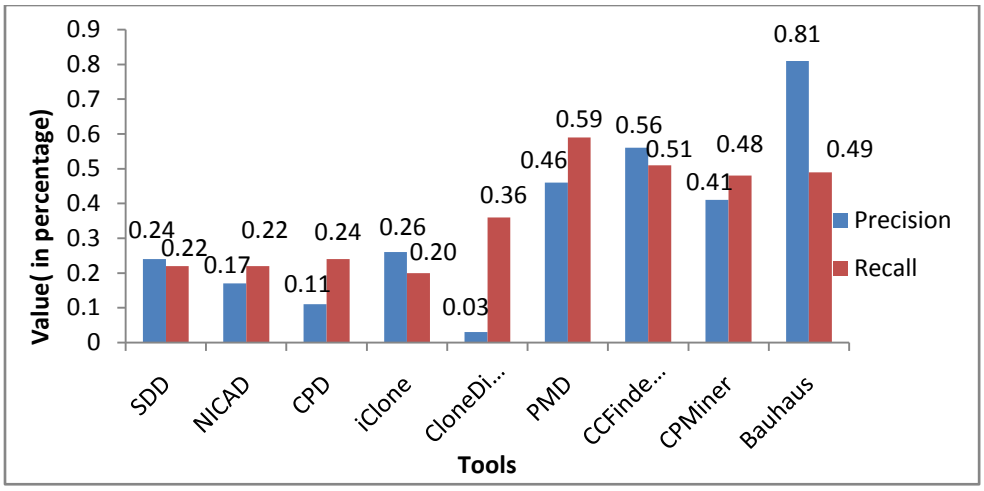


Figure 10 Comparisons of Precision and Recall Values of Clone Detection Tools

4.2PERFORMANCE EVALUATION FROM GENERATED TEST-CASES USING SOFTWARE METRICS

The clone detection tools have been compared by using the generated test-cases or software clone with the existing methodology which uses only two or three variants of software clones while we have evaluated detection tools by making various clones or test-cases which are created with the help of mutation-operators. Figure 11 shows the comparisons of precision of proposed methodology (computed using generated test-cases) with existing methodology.

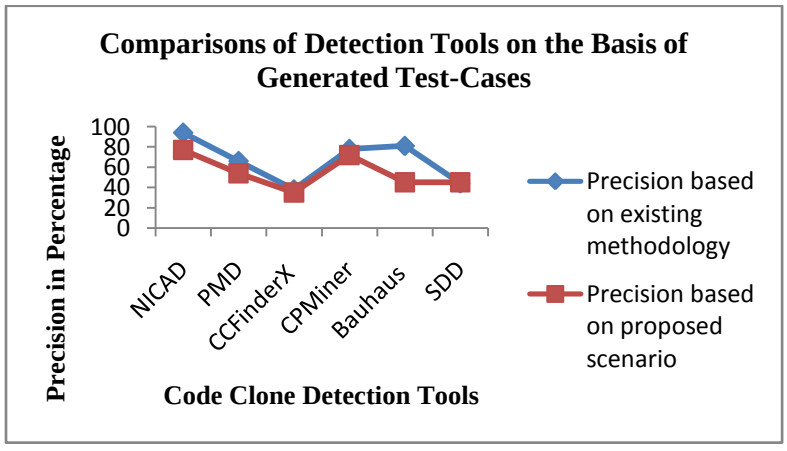


Figure 11 Performance Evaluations of Clone Detection Tools from Generated Test Cases with the Existing Methodology.

4.3 SUMMARY

In this chapter, we have proposed mutation operators in software cloning research area for generating various types of software clones. This chapter can be divided into three phases. In the first phase, we evaluate code clone detection tools by software metrics. In the second phase, we used mutation operators-based an editing taxonomy for making software clones and by utilizing these taxonomy variants, types of software clones will be created for type-1 to type-4. In the third phase, by using a mutation operator-based an editing taxonomy as well as various types of generated software clones used for developing a scenario. Further, the proposed scenario used for evaluating several clone detection tools and techniques on the basis of some effectiveness criteria as precision, recall, scalability, portability, and robustness.

5. USES OF MUTATION OPERATORS IN SOFTWARE CLONES

The empirical assessment of test techniques is a necessary part of software testing. The most proficient and vigorous software testing technique which is available is mutation testing. First time Lipton [42] introduced mutation testing in 1971 in a class paper and after that Hamlet [21] and DeMillo et al., [20] published a research paper on mutation testing. However, the mutation-testing used by the testing community over the past twenty five-years for testing the data. In mutation testing, bugs are introduced by creating several variants of source text [22]. Thus, we proposed mutation testing in software clone detection research field. Foremost, mutation operators used to generate various types of software clones as type-1 to type-3 code clones. Subsequent, by utilizing this mutation operator-based taxonomy a scenario has been developed so that on the basis of generated scenario the software clones detection tools and techniques can be evaluated empirically.

5.1 TOWARDS MUTATION TESTING-BASED EDITING TAXONOMY

The main concern for any mutation-based testing is the availability, of a set of defined mutation operators. The mutation-based testing method plays a vital role in evaluating software clone detection tools, if a set of mutation operators has been defined for a specified language(to

generate software clone) then it will be exemplary cloning in software clone detection research. Although, there are many mutation generators are available in various languages for generating probable bugs [43]. Thus, an editing nomenclature of mutation operator-based for creating distinct kinds of software clone has been proposed and by using that a set of mutation operators for cloning can be inferred. Moreover, assessing these mutations operators to introduce an evaluation structure that can be used to figure out clone detection tools using mutation operators.

5.2MUTATION OPERATOR-BASED SCENARIO FOR EVALUATING SOFTWARE CLONE DETECTION TOOLS AND TECHNIQUES

The software clone detection techniques and tools are based on their numerous attributes and sub-attributes which make them difficult to do the comparative study. Therefore, these tool and methods have been evaluated into two perspectives. Foremost, they were evaluated on the basis of their property and sub-property subsequently; a mutation operator-based taxonomy has been proposed for creating distinctive software clone and by using taxonomy, a scenario has been generated.

Although, Roy and Cordy [48 described) a hypothetical editing scenario for distinct clone types. Additionally, they also proposed a mutation-based control framework for assessing and comparing detection techniques which are based on that scenario. Moreover, no experiential research that has unambiguously inclined the application of mutants as far as our acquaintance is concerned. Thus, to overcome this issue, a hypothetical scenario is developed using mutation operator-based editing classification and this hypothetical scenario is used for evaluating software clone detection tools and techniques. Further, the existing evaluation criterion is extended by the hypothetical scenario which is presented by the analysis of results.

Figure 12 shows the mutation operator-based scenario which is created by mutation operator-based an editing taxonomy for evaluating of code clone detection tools and techniques.

Figure 13 depicts the text-based and token –based tools comparisons of proposed scenario with the existing scenario named as Roy’s scenario.

Figure 14 illustrates the tree-based, metric-based, and graph-based code clone detection tools comparisons of proposed scenario with the existing scenario named as Roy’s scenario.

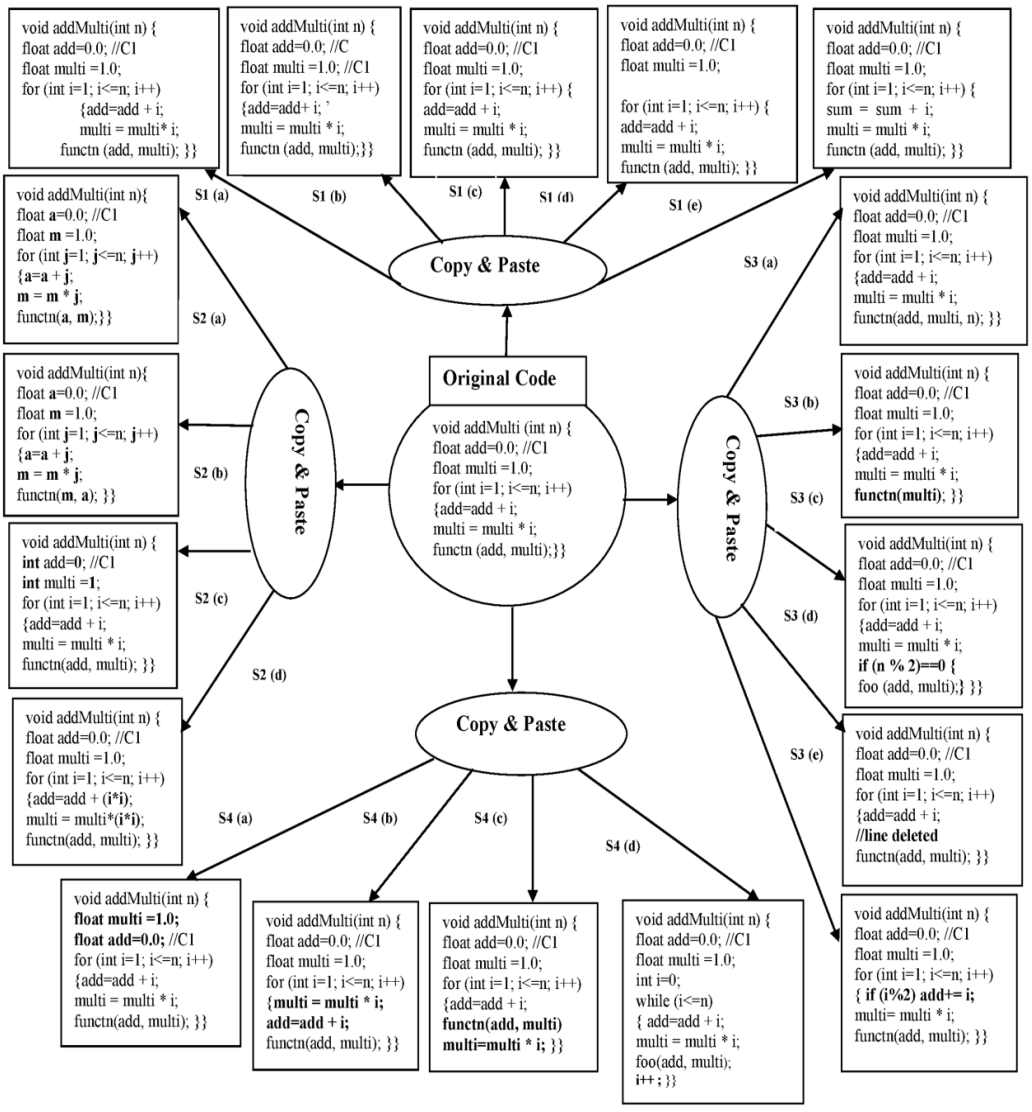


Figure 12 Mutation Operators- Based Editing Scenarios for Distinct Software Clone Types

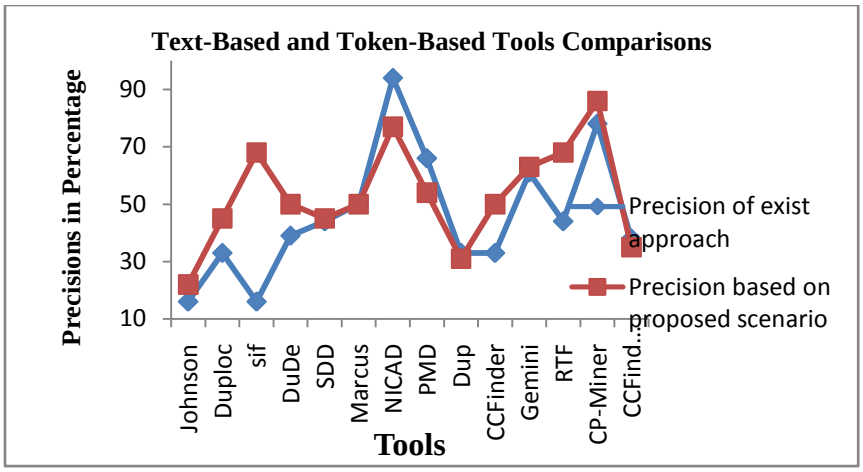


Figure 13 Scenario-Based Comparisons of String and Lexical-Based Approaches with Exist Scenario-Based and Proposed Scenario

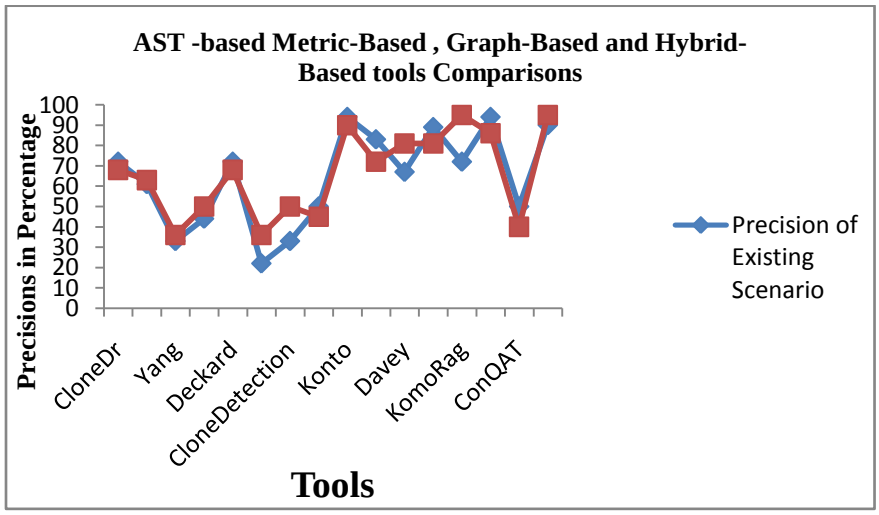


Figure 14 Scenario-Based Comparisons of Metrics, Tree and Graph-Based Approaches with Exist Scenario-Based and Proposed Scenario

5.3 AN AUTOMATIC FRAMEWORK FOR INJECTING AND DETECTING SOFTWARE CLONES

This chapter introduced an automatic-framework for evaluating software clone detection tools and techniques. The proposed framework is based on mutation testing which is a type of software testing in which we mutate (some alterations in source program) source code by using mutation operators. The mutation testing usually used by the testing community over the past

twenty-five years but in software clone detection field it is not used as per the author's best knowledge. Thus, the proposed method used mutation operators to generate variant software clones. The proposed framework is shown in figure 15.

The proposed framework consists of two phases of the clone generation phase and evaluation phase as shown in figure 15. The main aim of our abstraction is to outline a mutation-based valuation structure which has been used to evaluate clone detection tools and techniques. In the first phase random pieces of source code have been selected for generating mutants and after that by using mutation operators-based editing taxonomy software clones created for arbitrary source code. Further, these generated clones have been stored in the database. The second phase of proposed framework focus on detection of software clones. In clone evaluation phase the generated mutants of random source code have been replaced with the source code segments in the source code. In next step, a clone detection tool will be selected, and random mutant version of source code will be given as input into the detection tool for clone detection. The tools will be evaluated by that how many clones it will detect.

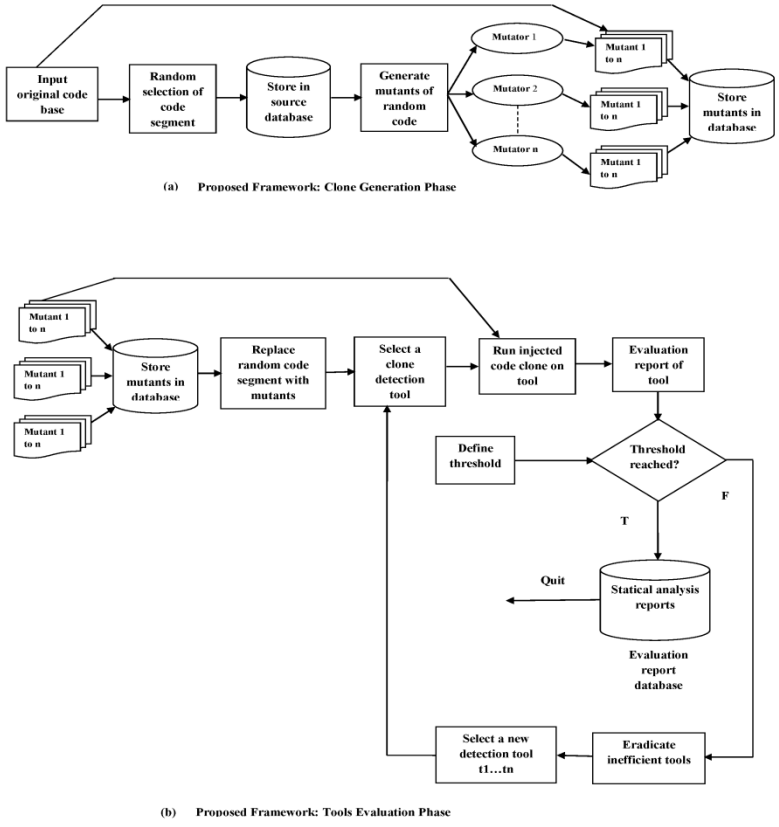


Figure 15 Proposed framework

5.4 SUMMARY

This chapter presents a blueprint of mutation operators-based framework for injection and evaluating software clone detection tools and techniques. The proposed structure is used to perform a qualitative evaluation of software clone detection and tools and techniques in term of precision and recall.

6. CONCLUSION AND FUTURE WORK

The Obfuscation is a security mechanism against reverse engineering, thus obfuscation's security is ensured at three different aspects of reverse engineering. The code clone is used to provide security to obfuscation against static and dynamic analysis attack in the proposed approach. Although, code clone used to protect code from the reverse engineering attack but on the other side code clone having a negative side too as it increases maintenance cost, execution speed, strain on system resources, bad impact on the design, and bug propagation [8]. Moreover, the software clones considered harmful in software evolution and maintenance because if a bug has been detected in one section of source code, then it requires corrections in the entire source code. In response, a few decades ago some code clone detection tools and techniques have been proposed [8]. The recent survey [9] reported that none of the tools and techniques had been introduced which can identify software clones with high precision and recall. Therefore, there is need to develop an approach which can detect code clones from the source code with high precision, recall, portability, scalability, and robustness. Thus, the proposed approach detects type-1 software clones [8] by a hybrid approach which has been discussed in chapter 3. The proposed method is based on lexical perception in which complete source code is partitioned into tokens, and it detects type-1 software clones (generated using mutation operators) with high precision and recalls and compares it with the existing method. By using directed acyclic graph (DAG), the type-2 software clones have been detected. The DAG is a compiler optimization technique which is used to removes common sub-expression from the source code. Further, non-trivial (functionally identical) code clone can be detected by the program dependency graph (PDG), and trivial software clones have been identified by the using control flow graph (CFG)

and reduced flow graph (RFG). Furthermore, the experimental valuation of software clone detection tools and techniques plays an imperative role in software testing and software clone detection research. Thus, there is need to evaluate the performance of existing code clone detection tools in terms of software metrics as precision, recall, portability, scalability, and robustness. Hence, the proposed approach evaluates software clone detection tools and techniques from two perspectives. Foremost, clone detection tools and techniques have been evaluated by recall, portability, and robustness subsequently, they evaluated by mutation testing-based generated test-cases which has been discussed in the chapter 4. Moreover, it is not mentioned in the literature how to generate code clones in the source code, and how to inject code clones in the source code. Thus, it is essential to propose a generic method for code clone creation, and proposed an automatic framework for code clone injection. Thus, the proposed approach is mentioned in the chapter 5. It uses mutation operators for generating variants of software clones by using an editing taxonomy which is based on mutation operators. The mutation-testing based scenario evaluates software clone detection tools and techniques by effectiveness criteria (precision, recall). Thus, there is a need to develop an approach which can detect trivial software clones (structurally similar) and non-trivial software clones (functionally identical). The software clones can be injected by hand or by using mutation operators [43]. However, the mutation operator generates software clones by editing activities, and several clone detection tools have been experimentally evaluated in the past. A lot of efforts have been done for empirically evaluating and analyzing various state-of-the-art tools. The existing study demonstrates that numerous traits that could leverage the authenticity of the outcome of such evaluation have been anticipated due to lack of legitimized software clone benchmark. Thus, to overcome this mutation testing-based automatic evaluation structure has been proposed for evaluating software clone detection tools and techniques. The proposed structure injects software clones automatically in the source code. However, the proposed structure is not completely implemented so in future it will be reported. In future, we will implement the proposed framework which can automatically inject software clones (generated using mutation operator-based taxonomy) as well as it can detect code clones automatically from type-1 to type-4 software clones.

REFERENCES

1. Tsipenyuk, K., Chess, B. and McGraw, G., 2005. Seven pernicious kingdoms: taxonomy of software security errors. *IEEE Security & Privacy*, *3*(6), pp.81-84.
2. Collberg, C.S. and Thomborson, C., 2002. Watermarking, tamper-proofing, and obfuscation-tools for software protection. *IEEE Transactions on software engineering*, *28*(8), pp.735-746.
3. Nagra, J. and Collberg, C., 2009. *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection: Obfuscation, Watermarking, and Tamperproofing for Software Protection*. Pearson Education.
4. Cappaert, J., 2012. Code obfuscation techniques for software protection. *Katholieke Universiteit Leuven*, pp.1-112.
5. Collberg, C., Thomborson, C. and Low, D., 1997. *A taxonomy of obfuscating transformations*. Department of Computer Science, The University of Auckland, New Zealand.
6. Kulkarni, A. and Lodha, S., 2012. Software protection through code obfuscation. *Project Report, Tata Research Development and Design Centre, Pune*.
7. Kulkarni, A. and Metta, R., 2014, April. A new code obfuscation scheme for software protection. In *Service Oriented System Engineering (SOSE), 2014 IEEE 8th International Symposium on* (pp. 409-414). IEEE Computer Society, Washington, DC, USA.
8. Rattan, D., Bhatia, R. and Singh, M., 2013. Software clone detection: A systematic review. *Information and Software Technology*, *55*(7), pp.1165-1199.
9. Roy, C.K. and Cordy, J.R., 2007. A survey on software clone detection research. *Queen's School of Computing TR*, *541*(115), pp.64-68.
10. Sheneamer, A. and Kalita, J., 2016. A survey of software clone detection techniques. *International Journal of Computer Applications*, *137*(10), pp.1-21.
11. Annual BSA global software piracy study," International Planning and Research Corporation, Tech. Rep., 2016. [Online]. Available: http://globalstudy.bsa.org/ 2016/downloads/studies/BSA GSS In Brief US.pdf
12. Gantz, J.F., Florean, A., Lee, R., Lim, V., Sikdar, B., Lakshmi, S.K.S., Madhavan, L. and Nagappan, M., 2014. The link between pirated software and cyber security breaches. *IDC White Paper Google Scholar*.
13. M. Fossi, G. Egan, K. Haley, E. Johnson, T. Mack, T. Adams, J. Blackbird, M. K.Low, D. Mazurek, D. McKinney *et al.*, "Symantec internet security threat report trends for 2010," *Volume XVI*, 2011.
14. Sajnani, H., Saini, V. and Lopes, C., 2015. A parallel and efficient approach to large scale clone detection. *Journal of Software: Evolution and Process*, *27*(6), pp.402-429.
15. Sajnani, H., Naik, R. and Lopes, C., 2011, October. Application Architecture Discovery-Towards Domain-driven, Easily-Extensible Code Structure. In *Reverse Engineering (WCRE), 2011 18th Working Conference on* (pp. 401-405). IEEE.
16. Roy, C.K., Cordy, J.R. and Koschke, R., 2009. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of computer programming*, *74*(7), pp.470-495.
17. Roy, C.K. and Cordy, J.R., 2009, April. A mutation/injection-based automatic framework for

evaluating code clone detection tools. In *IEEE International Conference on Software Testing, Verification, and Validation Workshops* (pp. 157-166). IEEE, *Computer Society*, Washington, DC, USA.
18. Roy, C.K. and Cordy, J.R., 2008, May. Towards a mutation-based automatic framework for evaluating code clone detection tools. In *Proceedings of the 2008 C 3 S 2 E conference* (pp. 137-140). ACM, New York, NY, USA.
19. Lipton, R.J., 1971. Fault diagnosis of computer programs. *Student Report, Carnegie Mellon University*.
20. DeMillo, R.A., Lipton, R.J. and Sayward, F.G., 1978. Hints on test data selection: Help for the practicing programmer. *Computer*, *11*(4), pp.34-41.
21. Hamlet, R.G., 1977. Testing programs with the aid of a compiler. *IEEE Transactions on Software engineering*, (4), pp.279-290.
22. Offutt, A.J., Lee, A., Rothermel, G., Untch, R.H. and Zapf, C., 1996. An experimental determination of sufficient mutant operators. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *5*(2), pp.99-118.
23. DeMillo, R.A., 1989, May. Completely validated software: test adequacy and program mutation (panel session). In *Proceedings of the 11th international conference on Software engineering* (pp. 355-356). ACM.
24. Woodward, M.R., 1993. Mutation testing—its origin and evolution. *Information and Software Technology*, *35*(3), pp.163-169.
25. Mathur, A.P., 2008. Foundations of software testing.
26. Ammann, P. and Offutt, J., 2016. *Introduction to software testing*. Cambridge University Press.
27. Offutt, J., 2011. A mutation carol: Past, present and future. *Information and Software Technology*, *53*(10), pp.1098-1107.
28. Ducasse, S., Rieger, M. and Demeyer, S., 1999. A language independent approach for detecting duplicated code. In *Software Maintenance, 1999. (ICSM'99) Proceedings. IEEE International Conference on* (pp. 109-118). IEEE, Oxford, UK.
29. Mayrand, J., Leblanc, C. and Merlo, E., 1996, November. Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics. In *icsm* (Vol. 96, p. 244).
30. Kontogiannis, K.A., DeMori, R., Merlo, E., Galler, M. and Bernstein, M., 1996. Pattern matching for clone and concept detection. *Automated Software Engineering*, *3*(1-2), pp.77-108.
31. Lague, B., Proulx, D., Mayrand, J., Merlo, E.M. and Hudepohl, J., 1997, October. Assessing the benefits of incorporating function clone detection in a development process. In *Software Maintenance, 1997. Proceedings., International Conference on* (pp. 314-321). IEEE, Washington, DC, USA.
32. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S. and Yang, K., 2012. On the (im) possibility of obfuscating programs. *Journal of the ACM (JACM)*, *59*(2), p.6.
33. Goldwasser, S. and Rothblum, G.N., 2014. On best-possible obfuscation. *Journal of Cryptology*, *27*(3), pp.480-505.
34. Schrittwieser, S., Katzenbeisser, S., Kinder, J., Merzdovnik, G. and Weippl, E., 2016. Protecting software through obfuscation: Can it keep pace with progress in code analysis?. *ACM Computing Surveys (CSUR)*, *49*(1), p.4.
35. Balakrishnan, A. and Schulze, C., 2005. Code obfuscation literature survey. *CS701 Construction of compilers*, *19*, Department of Computer Science, University of Wisconsin, Madison, US.
36. Ceccato, M., Di Penta, M., Nagra, J., Falcarin, P., Ricca, F., Torchiano, M. and Tonella, P.,

2009, May. The effectiveness of source code obfuscation: An experimental assessment. In *Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on* (pp. 178-187). IEEE.
37. Low, D., 1998. *Java control flow obfuscation* (Doctoral dissertation, University of Auckland).
38. Collberg, C., Thomborson, C. and Low, D., 1998, January. Manufacturing cheap, resilient, and stealthy opaque constructs. In *Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (pp. 184-196). ACM, New York, NY, USA.
39. Wang, C., Davidson, J., Hill, J. and Knight, J., 2001, July. Protection of software-based survivability mechanisms. In *Dependable Systems and Networks, 2001. DSN 2001. International Conference on* (pp. 193-202). IEEE, Washington, DC, USA.
40. Madou, M., Anckaert, B., De Sutter, B. and De Bosschere, K., 2005, November. Hybrid static-dynamic attacks against software protection mechanisms. In *Proceedings of the 5th ACM workshop on Digital rights management* (pp. 75-82). ACM, New York, NY, USA.
41. Cohen, F., 1987. Computer viruses. *Computers & security*, *6*(1), pp.22-35.
42. Lipton, R.J., 1971. Fault diagnosis of computer programs. *Student Report, Carnegie Mellon University*.
43. Andrews, J.H., Briand, L.C. and Labiche, Y., 2005, May. Is mutation an appropriate tool for testing experiments?. In *Proceedings of the 27th international conference on Software engineering* (pp. 402-411). ACM.

PUBLICATIONS LIST

Published Journal Papers

1. P. Gautam and H. Saini, "A Novel Software Protection Approach for Code Obfuscation to Enhance Software Security", *International Journal of Mobile Computing and Multimedia Communications (IJMCMC),*vol.8, no. 1, pp. 34-47, 2017.

   [Major indexing: SCOPUS, DBLP, ACM digital Library, web of sciences, Google scholar].

2. P. Gautam and H. Saini. "Non-Trivial Software Clone Detection Using Program Dependency Graph", *International Journal of Open Source Software and Procsses (IJOSSP),* vol. 8, no.2, pp. 1-24, 2017.

   [Major indexing: SCOPUS, DBLP, ACM digital Library, web of sciences, Google scholar].

3. P. Gautam and H. Saini, " A Mutation Operator-Based Scenario for Evaluating Software Clone Detection Tools and Techniques, *International Journal of Information Security and Privacy (IJISP),* vol. 13, no.1, 2017. (in press).

   [Major indexing: SCOPUS, ESCI, DBLP, ACM digital Library, web of sciences, Google scholar].

4. P. Gautam and H. Saini, "Correlation and Performance Estimation of Clone Detection Tools", *International Journal of Open Source Software and Processes (IJOSSP),* vol. 9, no. 2, 2018. (in press).

   [Major indexing: SCOPUS, DBLP, ACM digital Library, web of sciences, Google scholar].

Accepted Journal Papers

5. P. Gautam and H. Saini, "Mutation Testing-based Evaluation Framework for Evaluating Software Clone Detection Tools", *International Journal of Life Cycle Reliability and Safety Engineering*. Springer. vol. xx, no. xx, pp. xx-xx, 20xx.

[Major Indexing: *Google Scholar, CNKI, EBSCO Discovery Service, OCLC, Summon by ProQuest]*

Published Book Chapters

6. P. Gautam and H. Saini(2016), "Various Code Clone Detection Techniques and Tools: A Comprehensive Survey", *Proceedings of the International conference on Smart Trends for Information and Knowledge computing,* (CCIS'16), Springer, Singapore, pp. 255-267

   [Major indexing: SCOPUS, DBLP, Google scholar].

Published Conference Proceedings

7. P. Gautam and H. Saini (2017), "A Hybrid Approach for Detection of Type-1 Software Clones*", In 4th International conference on Signal Processing, Computing and Control (ISPCC-2017), IEEE,* Jaypee University of Information Technology, Waknaghat, Solan, HP, India, pp. 279-282.

[Major indexing: SCOPUS, Conference Proceedings Citation Index]

8. Pratiksha Gautam and Hemraj Saini(2018). "Type-2 Software Cone Detection Using Directed Acyclic Graph". *4th IEEE International conference on Image Information Processing (ICIIP - 2017)*, Jaypee University of Information Technology, Waknaghat, Solan, HP, India, (December 2017), pp. 205-208.

[Major indexing: SCOPUS, Conference Proceedings Citation Index]

Communicated Journal Papers

9. P. Gautam and H. Saini, "Mutation-Based Editing Taxonomy of Different Types of Software Clones", *International Journal of Information Technology and Web Engineering* [Under Review].

[Major indexing: SCOPUS, ESCI, DBLP, ACM digital Library, web of sciences, Google scholar].

10. P. Gautam and H. Saini, "Detection of Renamed Clones (type-2) Using Directed Acyclic Graph" *International Journal of Information Technology.* Springer, Communicated [Under Review].
[Major Indexing*: INSPEC, Google Scholar, CNKI, DBLP, EBSCO Discovery Service, OCLC, Proquest]*

11. P. Gautam and H. Saini (2018**). "**Detection of Type-1 Software Clones Using a Hybrid Approach" *In International Journal of Information Technology.* Springer, Communicated [Under Review].

[Major Indexing*: INSPEC, Google Scholar, CNKI, DBLP, EBSCO Discovery Service, OCLC, Proquest]*

12. P. Gautam and H. Saini. "Software Security Concerns with Open Research Issues — A Review", *In ACM Journal of computing survey* [Communicated].

[Major indexing: SCIE, SCOPUS, ESCI, DBLP, ACM digital Library, web of sciences, Google scholar].

Candidate's Signature